1. itertools.permutations(iterable[, r]) : This tool returns successive  length permutations of elements in an iterable. If  is not specified or is None, then  defaults to the length of the iterable, and all possible full length permutations are generated.Permutations are printed in a lexicographic sorted order. So, if the input iterable is sorted, the permutation tuples will be produced in a sorted order.
2. **from** itertools **import** permutations
3. S, k = **input**().split()
4. **print**('\n'.join(**sorted**(**map**(''.join,permutations(S,**int**(k))))))

map() function returns a map object(which is an iterator) of the results after applying the given function to each item of a given iterable (list, tuple etc.) Syntax :

map(fun, iter)

The map() function is used to apply this function to each element of the numbers list, and an if statement is used within the function to perform the necessary conditional logic.

* # Return double of n
* def addition(n):
* return n + n
* # We double all numbers using map()
* numbers = (1, 2, 3, 4)
* result = map(addition, numbers)
* print(list(result))

2. itertools.combinations(iterable, r)

This tool returns the length subsequences of elements from the input iterable. Combinations are emitted in lexicographic sorted order. So, if the input iterable is sorted, the combination tuples will be produced in sorted order.

* **from** itertools **import** combinations
* S, k  = **input**().split()
* **for** i **in** **range**(1, **int**(k)+1):
* comb = **list**(combinations(**sorted**(S),i))
* **for** each **in** comb:
* **print**("".join(each))

3. Reverse an array of integers.

* **def** reverseArray(a):
* *# method 1*
* **return** a[::-1]
* # method 2
  + rev = []
* **for** i **in** **range**(**len**(a)):
* rev.append(a[**len**(a)-1-i])
* **return** rev
* **if** **\_\_name\_\_** == '\_\_main\_\_':
* fptr = **open**(os.environ['OUTPUT\_PATH'], 'w')
* arr\_count = **int**(**input**().strip())
* arr = **list**(**map**(**int**, **input**().rstrip().split()))
* res = reverseArray(arr)
* fptr.write(' '.join(**map**(**str**, res)))
* fptr.write('\n')
* fptr.close()

LeetCode

4. Roman to interger

* class Solution:
* def romanToInt(self, s: str) -> int:
* roman\_to\_integer = {
* 'I': 1,
* 'V': 5,
* 'X': 10,
* 'L': 50,
* 'C': 100,
* 'D': 500,
* 'M': 1000,
* }
* num = 0
* s = s.replace("IV", "IIII").replace("IX", "VIIII").replace("XL", "XXXX").replace("XC", "LXXXX").replace("CD", "CCCC").replace("CM", "DCCCC")
* #return sum(map(lambda x: roman\_to\_integer[x], s))
* for i in s:
* num += roman\_to\_integer[i]
* return num

1. Your task is to find out if the string contains: alphanumeric characters, alphabetical characters, digits, lowercase and uppercase characters.

**if** **\_\_name\_\_** == '\_\_main\_\_':

    s = **input**()

    methods = ["isalnum", "isalpha", "isdigit", "islower", "isupper"]

**for** i **in** methods:

        case = **False**

**for** k **in** s:

**if** **getattr**(k, i)():

                case = **True**

**break**

**print**(case)

Python getattr() function is used to access the attribute value of an object and also gives an option of executing the default value in case of unavailability of the key.

Python any() Function: The any() function returns True if any item in an iterable are true, otherwise it returns False. If the iterable object is empty, the any() function will return False.

* print (any(s.isalnum()for s in s))
* print (any(s.isalpha()for s in s))
* print (any(s.isdigit()for s in s))
* print (any(s.islower()for s in s))
* print (any(s.isupper()for s in s))
* return

Que. 1768. Merge Strings Alternately

Easy

You are given two strings word1 and word2. Merge the strings by adding letters in alternating order, starting with word1. If a string is longer than the other, append the additional letters onto the end of the merged string.

Return the merged string.

* class Solution:
* def mergeAlternately(self, x: str, y: str) -> str:
* res = ""
* mn = min(len(x), len(y))
* mx = max(len(x), len(y))
* for i in range(mn):
* res += x[i]
* res += y[i]
* if len(x) > len(y):
* res += x[mn:]
* else:
* res += y[mn:]
* return res

Que 1071. Greatest Common Divisor of Strings

For two strings s and t, we say "t divides s" if and only if s = t + ... + t (i.e., t is concatenated with itself one or more times).

Given two strings str1 and str2, return the largest string x such that x divides both str1 and str2.

* class Solution:
* def gcdOfStrings(self, str1: str, str2: str) -> str:
* len1, len2 = len(str1), len(str2)
* def valid(k):
* if len1 % k or len2 % k:
* return False
* n1, n2 = len1 // k, len2 // k
* base = str1[:k]
* return str1 == n1 \* base and str2 == n2 \* base
* for i in range(min(len1, len2), 0, -1):
* if valid(i):
* return str1[:i]
* return ""

Que. 1431. Kids With the Greatest Number of Candies

There are n kids with candies. You are given an integer array candies, where each candies[i] represents the number of candies the ith kid has, and an integer extraCandies, denoting the number of extra candies that you have.

Return a boolean array result of length n, where result[i] is true if, after giving the ith kid all the extraCandies, they will have the greatest number of candies among all the kids, or false otherwise.

Note that multiple kids can have the greatest number of candies.

* class Solution:
* def kidsWithCandies(self, candies: List[int], extraCandies: int) -> List[bool]:
* maxn = max(candies)
* li = list(map(lambda x : x + extraCandies,candies))
* output = list(map(lambda x : True if x >= maxn else False,li))
* return output
* class Solution:
* def kidsWithCandies(self, candies: List[int], extraCandies: int) -> List[bool]:
* maxn = max(candies)
* li = list(map(lambda x : True if (x + extraCandies) >= maxn else False,candies))
* return li

Que. 605. Can Place Flowers. You have a long flowerbed in which some of the plots are planted, and some are not. However, flowers cannot be planted in adjacent plots.

Given an integer array flowerbed containing 0's and 1's, where 0 means empty and 1 means not empty, and an integer n, return true if n new flowers can be planted in the flowerbed without violating the no-adjacent-flowers rule and false otherwise.

* class Solution:
* def canPlaceFlowers(self, flowerbed: List[int], n: int) -> bool:
* length = len(flowerbed)
* count0 = flowerbed.count(0)
* if count0 >= (n+2):
* return True
* else:
* return False

Que 605. Can Place Flowers

You have a long flowerbed in which some of the plots are planted, and some are not. However, flowers cannot be planted in adjacent plots.

Given an integer array flowerbed containing 0's and 1's, where 0 means empty and 1 means not empty, and an integer n, return true if n new flowers can be planted in the flowerbed without violating the no-adjacent-flowers rule and false otherwise.

* class Solution:
* def canPlaceFlowers(self, flowerbed: List[int], n: int) -> bool:
* a = 0
* for i in range(len(flowerbed)):
* if flowerbed[i]==0 and ((i == len(flowerbed) - 1) or (flowerbed[i + 1] == 0)) and ((i == 0) or (flowerbed[i - 1] == 0)) :
* flowerbed[i] = 1
* a += 1
* if a >= n:
* return True
* else:
* return False